

## 1、Ajax特点

优点：

- 可以无需刷新页面与服务器进行通信
- 允许根据用户事件来更新部分页面内容

缺点：

- 没有浏览记录，不能回退
- 存在跨域问题
- SEO不友好（爬虫获取不到Ajax请求的数据）

### get与post的区别：

1. get请求一般是去获取数据(其实也可以提交,但常见的是获取数据); post请求一般是去提交数据。
2. get因为参数会放在url中, 所以隐私性, 安全性较差, 请求的数据长度是有限制的, 不同的浏览器和服务器的不同, 一般限制在 2~8K 之间, 更加常见的是 1k 以内; post请求是没有的长度限制, 请求数据是放在body中;
3. get请求刷新服务器或者回退没有影响, post请求回退时会重新提交数据请求。
4. get请求可以被缓存, post请求不会被缓存。

## 2、HTTP协议

### 1、请求报文

#### 1、请求行

类型：get post

URL路径

HTTP协议版本

#### 2、请求头

格式：

- Host: baidu.com
- Cookie: name=guang
- Content-type: application/x-www-form-urlencoded
- User-Agent: chrome 83

#### 3、请求空行

必须要有

#### 4、请求体

- get请求：请求体为空
- post请求：可为空可不为空，eg: username=guang&password=123456

## 2、响应报文

## 1. 响应行

- HTTP协议版本
- 响应状态码：200、404、403、500、401等
- 响应状态字符串：和状态码相同

## 2. 响应头

格式：

- Content-Type: text/html; charset=utf8
- Content-length: 2048
- Content-encoding: gzip

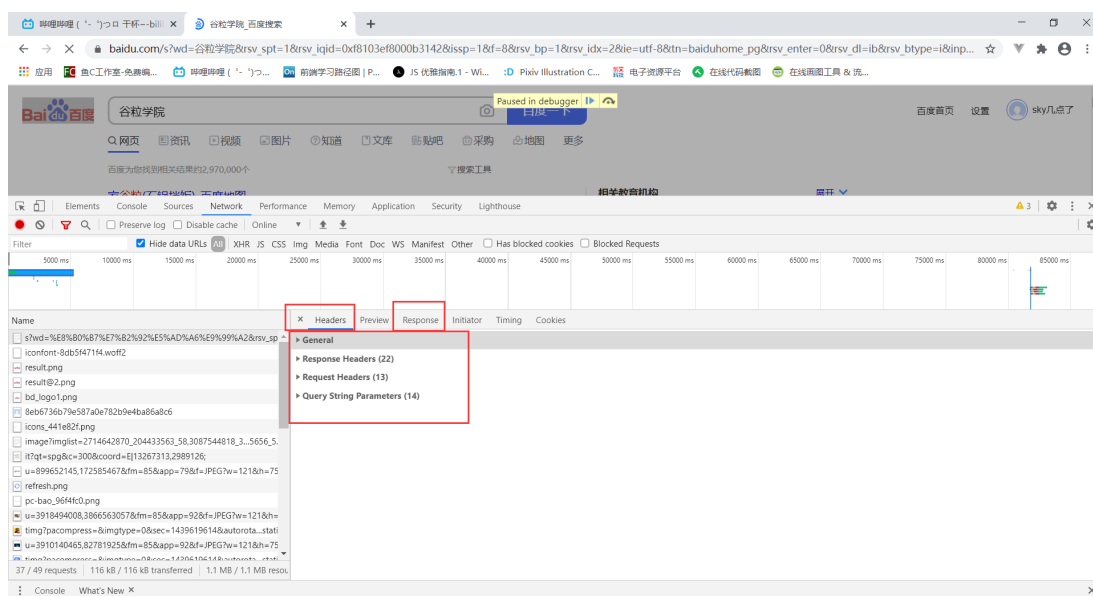
## 3. 响应空行

必须要有

## 4. 响应体

响应主要内容

## 3、get请求演示

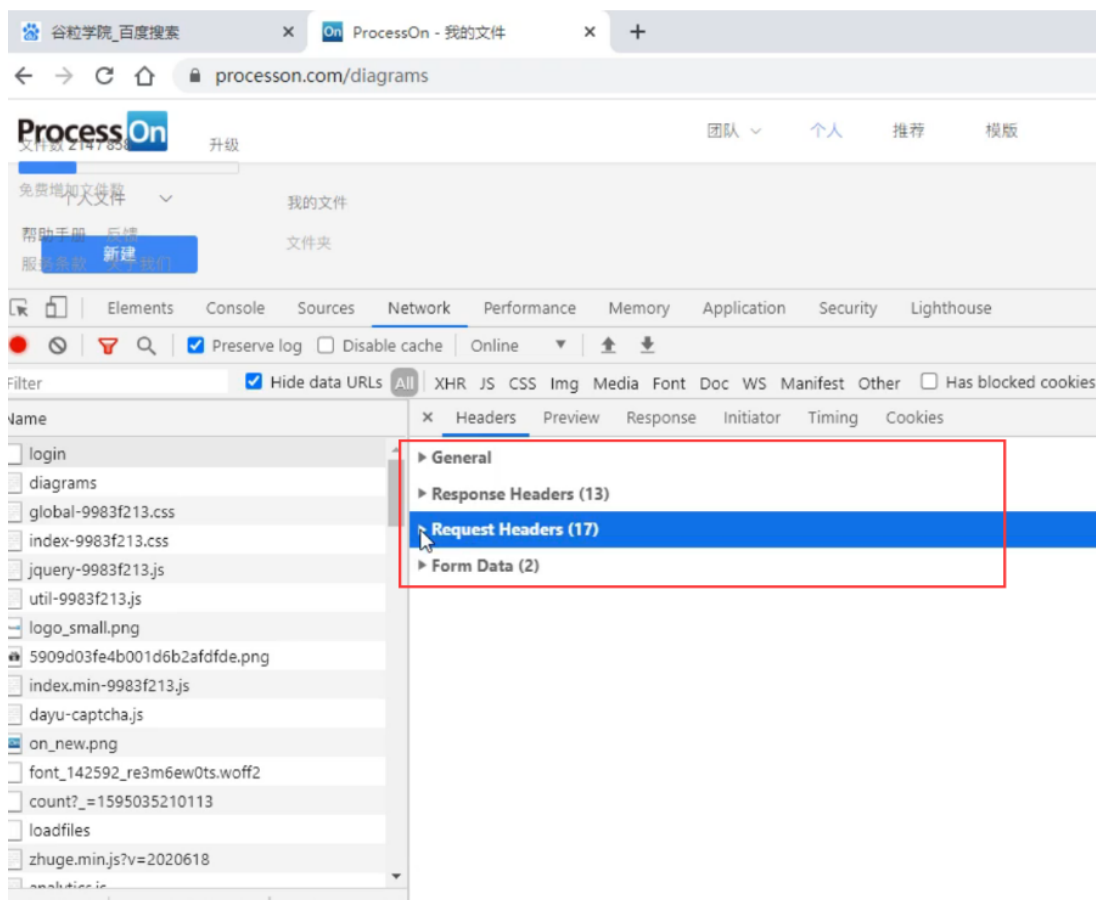


Headers:

- General
- Request Headers为请求头信息
- Response Headers为响应头信息
- Query String Parameters查询字符串参数:对url参数的解析

Response为服务端返回的html内容

## 4、post请求演示



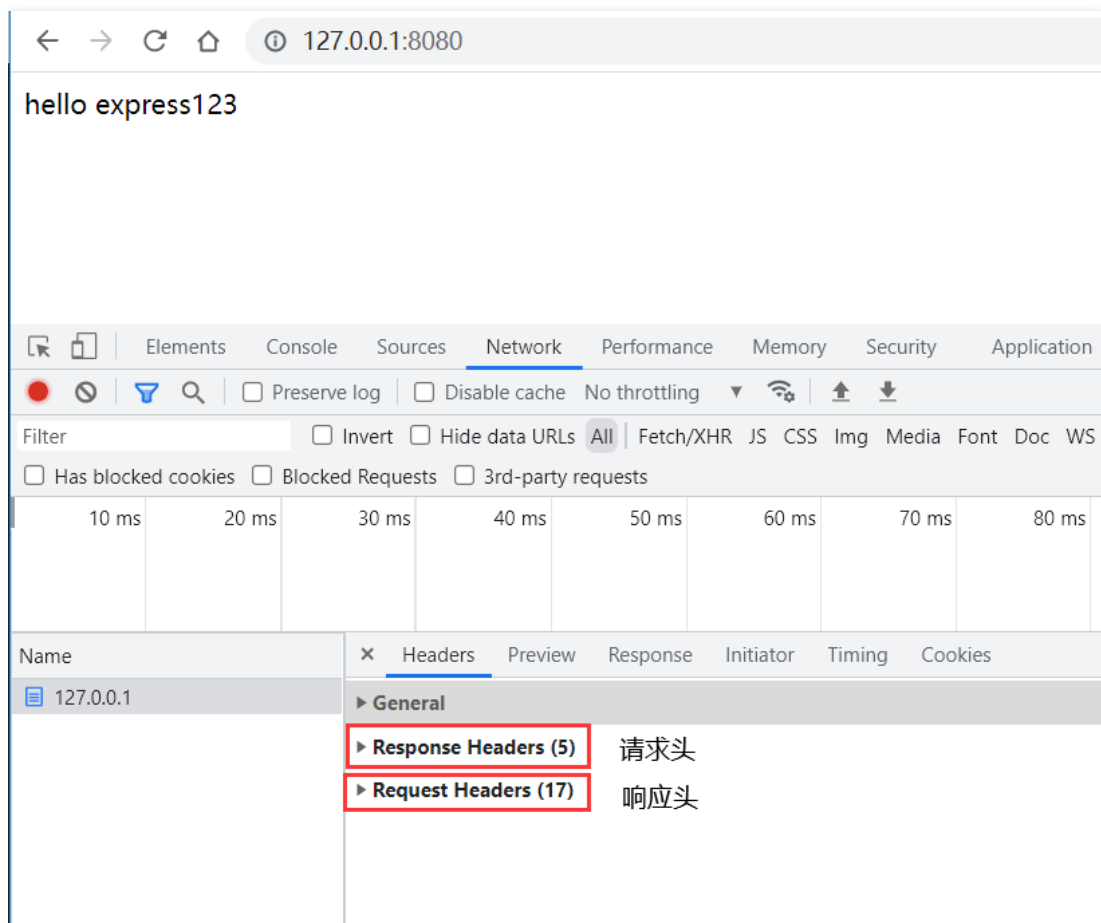
Form Data为请求体的内容

### 3、原生Ajax

#### 1 express基本使用



通过node.js启动后可以在127.0.0.1:8080访问



## 2 ajax请求基本步骤

get请求

html

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>...
4 </head>
5 <body>
6     <!-- 需求: 点击发送请求, 将相应信息返回显示至ajax中 页面不刷新 -->
7     <button>点击发送请求</button>
8     <div id="res"></div>
9     <script>
10         const btn = document.getElementsByTagName('button')
11         const result = document.getElementById('res')
12         btn[0].onclick = function () {
13             //创建对象
14             const xhr = new XMLHttpRequest()
15             //设置请求方法和url
16             xhr.open('GET', 'http://localhost:8000/server')
17             //发送
18             xhr.send()
19             //事件绑定 处理服务端返回的结果
20             xhr.onreadystatechange = function () {
21                 if(xhr.readyState === 4){ //服务端返回了所有结果
22                     //2开头的都表示成功
23                     if(xhr.status >= 200 && xhr.readyState<=300){
24                         //处理结果
25                         console.log(xhr.status) //状态码
26                         console.log(xhr.statusText) //状态字符串
27                         console.log(xhr.getAllResponseHeaders) //所有响应头
28                         console.log(xhr.response) //响应体
29                         result.innerHTML = xhr.response
30                     }else{}
31                 }
32             }
33         }
34     </script>
35 </body>
36 </html>

```

```

1 <button>点击发送请求</button>
2     <div id="result"></div>
3     <script>
4         //1
5         const btn = document.getElementsByTagName('button')[0];
6         const result = document.getElementById('result');
7         //2
8         btn.onclick = function () {
9             //1 创建XMLHttpRequest对象
10            const xhr = new XMLHttpRequest();
11            //2 初始化: 用何种方式发请求? 给谁发请求? 带着什么参数过去?

```

```

12     xhr.open('GET', 'http://localhost:8000/server?a=100&b=200&c=300');
13     //3 发送
14     xhr.send();
15     //4 事件绑定
16     //on when 当...时候
17     //xhr内部有5种状态，标识着当前xhr内部读取数据的状态,分别是:
18     //0: 开头 表示未初始化
19     //1: 开头 表示open方法调用完毕
20     //2: 开头 表示send方法调用完毕
21     //3: 开头 表示服务端返回部分结果
22     //4: 开头 表示服务端返回所有结果
23
24     //5 处理服务端返回结果
25     xhr.onreadystatechange = function () {
26         //判断（服务端返回了所有结果）
27         if (xhr.readyState === 4) {
28             //判断状态码200 404 403 401 500
29             //2xx 成功 2开头的都表示成功
30             if (xhr.status >= 200 && xhr.status < 300) {
31                 //处理结果 行 头 空行 体
32                 //响应行
33                 console.log(xhr.status); //状态码
34                 console.log(xhr.statusText); //状态字符串
35                 console.log(xhr.getAllResponseHeaders()); //所有响应头
36                 console.log(xhr.response); //响应体
37                 //设置 result 文本
38                 result.innerHTML = xhr.response;
39             } else {}
40         }
41     }
42 }
43 </script>

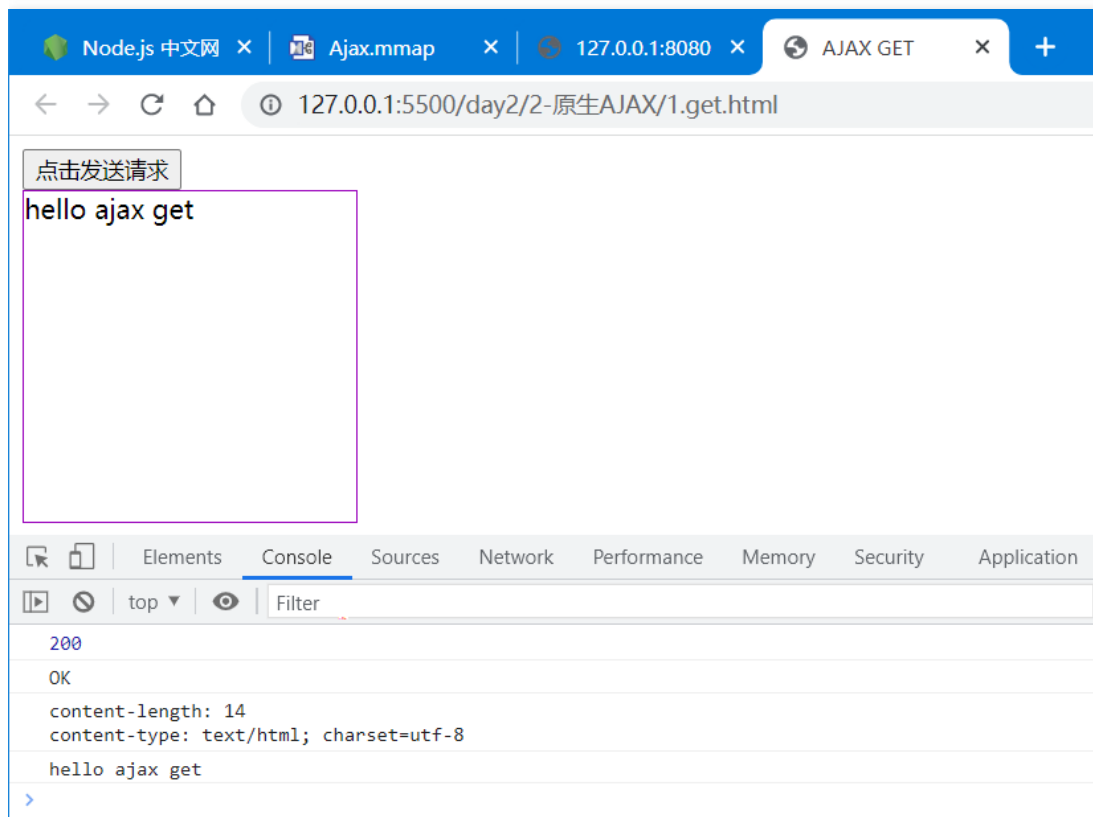
```

服务端：

```

1 //引入express并创建应用对象
2 const express = require('express')
3 const app = express()
4
5 //创建路由规则
6 app.get('/server', (request, response)=>{
7     //设置响应头 设置允许跨域
8     response.setHeader('Access-Control-Allow-Origin', '*')
9     //设置响应
10    response.send('hello ajax')
11 })
12
13 //监听端口启动服务
14 app.listen(8000, ()=>{
15     console.log('服务已经启动, 端口8000监听中')
16 })
17

```



### 3、post请求

代码演示:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>...
4 </head>
5 <body>
6   <div id="res"></div>
7   <script>
8     const result = document.getElementById('res')
9     result.onmouseover = function(){
10       const xhr = new XMLHttpRequest()
11       //与上一个GET请求相比，只变化了这一部分
12       xhr.open('POST', 'http://localhost:8000/server')
13       xhr.send()
14       xhr.onreadystatechange = function(){
15         if(xhr.readyState === 4){
16           if(xhr.status >= 200 && xhr.status < 300){
17             result.innerHTML = xhr.response
18           }
19         }
20       }
21     }
22   </script>
23 </body>
24 </html>

```

```

1 //引入express并创建应用对象
2 const express = require('express')
3 const app = express()
4
5 //创建路由规则
6 app.get('/server', (request, response) => {
7   //设置响应头 设置允许跨域
8   response.setHeader('Access-Control-Allow-Origin', '*')
9   //设置响应
10  response.send('hello ajax')
11 })
12 //接受post请求
13 app.post('/server', (request, response) => {
14   //设置响应头 设置允许跨域
15   response.setHeader('Access-Control-Allow-Origin', '*')
16   //设置响应
17   response.send('hello ajax post')
18 })
19
20 //监听端口启动服务
21 app.listen(8000, () => {
22   console.log('服务已经启动，端口8000监听中')
23 })

```

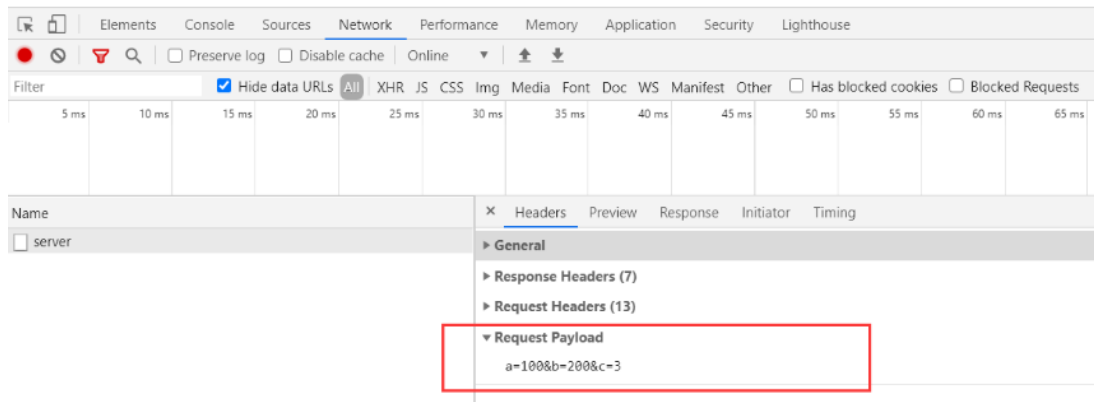
## post请求体参数设置:

位置: 设置在 xhr.send()内

格式: 'a=100&b=200&c=300' 或 'a:100&b:200&c:2'



hello ajax post



### 设置请求头:

- 专用方法: `xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')`
- 参数: 接收两个参数, 头的名字和头的值 同样可以在network中查看

```
1 <body>
2   <div id="result"></div>
3   <script>
4     //1
5     const result = document.querySelector('#result');
6     //2
7     result.addEventListener('click', function () {
8       //console.log('test');
9       const xhr = new XMLHttpRequest();
10      xhr.open('post', 'http://127.0.0.1:8000/server');
11
12      //设置请求头
13      xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
14      xhr.setRequestHeader('name', 'atguigu');
15
16      xhr.send('a=100&b=200&c=300');
17      xhr.onreadystatechange = function () {
18        if (xhr.readyState === 4) {
19          if (xhr.status >= 200 && xhr.status < 300) {
20            result.innerHTML = xhr.response;
21          }
22        }
23      }
24    });
25  </script>
26 </body>
```

```
23     }
24   })
25 </script>
26 </body>
```

#### 4、服务端响应json数据

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>...
4 </head>
5 <body>
6   <div id="res"></div>
7   <script>
8     const result = document.getElementById('res')
9     window.onkeydown = function(){
10       const xhr = new XMLHttpRequest()
11       //设置响应体数据的类型
12       xhr.responseType = 'json'
13       xhr.open('GET', 'http://localhost:8000/json-server')
14       xhr.send()
15       xhr.onreadystatechange = function(){
16         if(xhr.readyState === 4){
17           if(xhr.status >= 200 && xhr.status < 300){
18             console.log(xhr.response)
19
20             //手动对数据进行转换
21             // let data = JSON.parse(xhr.response)
22             // console.log(data)
23
24             //自动转换 上部须设置响应体数据类型
25             console.log(xhr.response)
26             result.innerHTML = xhr.response.name
27           }
28         }
29       }
30     }
31   </script>
32 </body>
33 </html>
```

```

1 //引入express并创建应用对象
2 const express = require('express')
3 const app = express()
4
5 //创建路由规则
6 app.get('/server', (request, response)=>{...
7 })
8 app.post('/server', (request, response)=>{...
9 })
10
11 //创建路由规则
12 app.get('/json-server', (request, response)=>{
13     //设置响应头 设置允许跨域
14     response.setHeader('Access-Control-Allow-Origin', '*')
15     //相应一个数据
16     const data = {
17         name: 'hanser'
18     }
19     //对对象进行字符串转化
20     let str = JSON.stringify(data)
21     //设置响应
22     response.send(str)
23 })
24 //监听端口启动服务
25 app.listen(8000, ()=>{
26     console.log('服务已经启动, 端口8000监听中')
27 })
28

```

## 5、Ajax请求超时与网络异常处理

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>AJAX 超时处理</title>
9     <style>
10         #result {
11             width: 200px;
12             height: 200px;
13             border: 1px solid pink;
14         }
15     </style>

```

```

16 </head>
17
18 <body>
19     <button>点击</button>
20     <div id="result"></div>
21     <script>
22         const btn = document.querySelector('button');
23         const result = document.querySelector('#result');
24
25         result.addEventListener('click', function () {
26             const xhr = new XMLHttpRequest();
27             //超时设置
28             xhr.timeout = 2000;
29             //超时回调
30             xhr.ontimeout = function () {
31                 alert('网络异常，请稍后再试')
32             }
33             //网络异常回调
34             xhr.onerror = function () {
35                 alert('你的网络似乎出了问题')
36             }
37
38             xhr.open('get', 'http://127.0.0.1:8000/delay');
39             xhr.send('');
40             xhr.onreadystatechange = function () {
41                 if (xhr.readyState === 4) {
42                     if (xhr.status >= 200 && xhr.status < 300) {
43                         result.innerHTML = xhr.response;
44                     }
45                 }
46             }
47         })
48     </script>
49 </body>
50
51 </html>

```

## 6、取消请求

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>AJAX 取消请求</title>
9 </head>
10
11 <body>
12     <button>点击发送</button>
13     <button>点击取消</button>
14     <script>
15         const btns = document.querySelectorAll('button')
16         let xhr = null;
17         btns[0].onclick = function(){
18             xhr = new XMLHttpRequest()
19             xhr.open('GET', 'http://localhost:8000/delay')
20             xhr.send();
21         }
22         //取消请求的回调函数 注意需要将xhr放在外侧
23         btns[1].onclick = function(){
24             xhr.abort();
25         }
26     </script>
27 </body>
28
29 </html>
```

## 7、解决请求重复发送的问题

如果连续发送同样的请求，会给服务器造成很大的不必要的压力。

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
```

```

6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>AJAX 重复请求</title>
9 </head>
10
11 <body>
12     <button>点击发送</button>
13     <script>
14         const btns = document.querySelectorAll('button');
15         let xhr = null;
16         //标识变量
17         let flag = false;
18         btns[0].onclick = function () {
19             //判断标识变量
20             if (flag) xhr.abort();
21             xhr = new XMLHttpRequest();
22             //修改标识变量的值
23             flag = true;
24             xhr.open('GET', 'http://localhost:8000/delay');
25             xhr.send();
26             xhr.onreadystatechange = function () {
27                 //readyState=4: 响应的内容解析完毕，可以在客户端使用了--完成
28                 if (xhr.readyState === 4) {
29                     //改变标识变量
30                     flag = false;
31                 }
32             }
33         }
34     </script>
35 </body>
36
37 </html>

```

server:

```

1 //1 引入express
2 const express = require('express');
3
4 //2 创建应用对象

```

```
5  const app = express();
6
7  //3 创建路由规则
8  //get请求
9  app.get('/server', (request, response) => {
10      //设置响应头 设置允许跨域
11      response.setHeader('Access-Control-Allow-Origin', '*');
12      //设置响应体
13      response.send('hello ajax get');
14  });
15
16  //post请求
17  app.post('/server', (request, response) => {
18      //设置响应头 设置允许跨域
19      response.setHeader('Access-Control-Allow-Origin', '*')
20      //设置响应
21      response.send('hello ajax post')
22  });
23
24  //JSON响应
25  app.all('/json-server', (request, response) => {
26      //设置响应头 设置允许跨域
27      response.setHeader('Access-Control-Allow-Origin', '*');
28      //响应头
29      response.setHeader('Access-Control-Allow-Headers', '*');
30      //响应一个数据
31      const data = {
32          name: 'guang'
33      };
34      //对对象进行字符串转化
35      let str = JSON.stringify(data);
36      //设置响应体
37      response.send(str);
38  });
39
40  //延时响应
41  app.all('/delay', (request, response) => {
42      //设置响应头 设置允许跨域
43      response.setHeader('Access-Control-Allow-Origin', '*');
44      setTimeout(() => {
```

```
45     //设置响应体
46     response.send('延时响应');
47 }, 3000)
48 });
49
50 //4 监听端口启动服务
51 app.listen(8000, () => {
52     console.log('服务器启动了, 8000端口监听中...');
53 });
```

## 4、axios

### 1 基本使用

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>axios发送ajax请求</title>
8     <style>
9         button {
10             border-radius: 5px;
11             height: 35px;
12             width: 100px;
13             background-color: blueviolet;
14             float: left;
15             margin-right: 10px;
16             color: white;
17             border: none;
18             text-align: center;
19         }
20
21         .btn-primary {
22             margin-left: 160px;
23             background-color: #cd6969;
24         }
25
```



```
26     .btn-danger {
27         background-color: orange;
28     }
29 </style>
30 <script src="https://cdn.bootcdn.net/ajax/libs/axios/0.19.2/axios.js"></script>
31 </head>
32
33 <body>
34     <button class="btn btn-primary">GET</button>
35     <button class="btn btn-danger">POST</button>
36     <button class="btn btn-info">通用型方法</button>
37     <script>
38         const btns = document.querySelectorAll('button')
39         //1 get请求
40         //第一个参数为url地址，第二个为其他参数配置的对象
41         btns[0].onclick = function () {
42             axios.get('http://localhost:8000/axios-server', {
43                 //url参数
44                 params: {
45                     id: 1000,
46                     vip: 12
47                 },
48                 //请求头信息
49                 headers: {
50                     name: 'hanser',
51                     age: '2'
52                 }
53             }).then(value => {
54                 console.log(value)
55             })
56         }
57         //2 post请求
58         //第一个参数为url地址，第二个为请求体，第三个为其他参数配置
59         btns[1].onclick = function () {
60             axios.post('http://localhost:8000/axios-server', {
61                 username: 'admin',
62                 password: '123'
63             }, {
64                 //url参数
65                 params: {
```

```
66         id: 1,
67         vip: 123
68     },
69     //请求头参数
70     headers: {
71         name: 'yousa',
72         age: '23'
73     },
74     //请求体
75 })
76 }
77 //3 通用方式发送
78 //参数只有一个对象，参数顺序与报文顺序一致
79 btns[2].onclick = function () {
80     axios({
81         method: 'POST',
82         url: 'http://localhost:8000/axios-server',
83         //url参数
84         params: {
85             vip: 10,
86             id: 123
87         },
88         //头信息
89         headers: {
90             a: 100,
91             b: 200
92         },
93         //请求体参数
94         data: {
95             name: 'hanser',
96             age: '7'
97         }
98     }).then(response => {
99         console.log(response)
100     })
101 }
102 </script>
103 </body>
104 </html>
```

server:

```
1 //1 引入express
2 const express = require('express');
3
4 //2 创建应用对象
5 const app = express();
6
7 //3 创建路由规则
8 // axios 服务
9 app.all('/axios-server', (request, response) => {
10     //设置响应头 设置允许跨域
11     response.setHeader('Access-Control-Allow-Origin', '*')
12     response.setHeader('Access-Control-Allow-Headers', '*')
13     const data = { name: 'hanser' }
14     //设置响应
15     response.send(JSON.stringify(data))
16 })
17
18 //4 监听端口启动服务
19 app.listen(8000, () => {
20     console.log('服务器启动了, 8000端口监听中...');
21 });
```

## 2、fetch函数发送ajax请求

```

1 <script>
2   const btn = document.getElementsByTagName('button')
3   btn[0].onclick = function(){
4     fetch('http://localhost:8000/fetch-server',{
5       //请求方法
6       method:'POST',
7       //请求头
8       headers:{
9         name:'hanser'
10      },
11      //请求体
12      body: 'name=admin&pd=admin'
13    }).then(Response=>{
14      console.log(Response)
15      return Response.text()
16    }).then(Response=>{
17      console.log(Response)
18    })
19  }
20 </script>

```

server:

```

1 //1 引入express
2 const express = require('express');
3
4 //2 创建应用对象
5 const app = express();
6
7 //3 创建路由规则
8 // fetch 服务
9 app.all('/fetch-server', (request, response) => {
10   //设置响应头 设置允许跨域
11   response.setHeader('Access-Control-Allow-Origin', '*')
12   response.setHeader('Access-Control-Allow-Headers', '*')
13
14   const data = { name: 'hanser' }
15   //设置响应
16   response.send(JSON.stringify(data))
17 })
18
19 //4 监听端口启动服务
20 app.listen(8000, () => {
21   console.log('服务器启动了, 8000端口监听中...');

```

## 5、跨域

### 1 Ajax同源策略

同源策略是浏览器的一种安全策略，违背同源策略就是跨域

同源:协议，域名，端口号必需完全相同

### 2 如何解决跨域

jsonp解决跨域问题：

ajax响应的返回结果是函数调用，而函数的实参就是我们想给客户端返回的结果数据，而这个函数必须于线定义好

思路就是往文件里添加script标签，标签引入内容的格式为上方所述

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>跨域</title>
8 </head>
9
10 <body>
11     <h1>hanser</h1>
12     <button>点击获取用户数据</button>
13     <script>
14         const btn = document.getElementsByTagName('button')
15         btn[0].onclick = function () {
16             const x = new XMLHttpRequest()
17             //因为满足同源策略，所以url可以简写
18             x.open('GET', '/data')
19             x.send()
20             x.onreadystatechange = function () {
21                 if (x.readyState === 4) {
22                     if (x.status >= 200 && x.status < 300) {
23                         console.log(x.response)
24                     }
25                 }
26             }
27         }
28     </script>
29 </body>
30 </html>
```

```
25         }
26     }
27 }
28 </script>
29 </body>
30
31 </html>
```

服务端：

```
1  const express = require('express')
2
3  const app = express()
4  app.get('/home', (request, response) => {
5      response.sendFile(__dirname + '/index.html')
6  })
7
8  app.get('/data', (request, response) => {
9      response.send('用户数据')
10 })
11
12 app.listen(9000, () => {
13     console.log('端口9000服务已经启动')
14 })
```

## 6 取消请求

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>练习取消请求</title>
6  </head>
7  <body>
8
9      <button id="btn">点我获取验证码</button>
10     <div id="test"></div>
11
```

```

12 <script type="text/javascript">
13     let btn = document.getElementById('btn')
14     let lastXhr
15     btn.onclick = function () {
16         if (lastXhr) lastXhr.abort()
17         lastXhr = getCode()
18     }
19
20     function getCode() {
21         let xhr = new XMLHttpRequest()
22         xhr.onreadystatechange = function () {
23             if (xhr.readyState === 4 && xhr.status === 200) {
24                 console.log('验证码是: ', xhr.response)
25             }
26         }
27         xhr.open('get', 'http://localhost:3000/get_verify_code')
28         xhr.send()
29         return xhr
30     }
31 </script>
32
33 </body>
34 </html>

```

server:

```

1 //引入express
2 const express = require('express')
3
4 //实例一个app应用对象
5 const app = express()
6
7 //使用内置中间件去解析post请求中以urlencoded形式编码的参数
8 app.use(express.urlencoded({ extended: true }))
9
10 //暴露静态资源
11 app.use(express.static(__dirname + '/public'))
12
13 app.get('/get_verify_code', (req, res) => {

```

```

14 console.log('有人找我要验证码了')
15 setTimeout(() => {
16     let code = Math.floor(Math.random() * 8999 + 1000)
17     res.send(code.toString())
18 }, 1000)
19 })
20
21 //
22 app.listen(3000, (err) => {
23     if (err) console.log(err)
24     else {
25         console.log('不要用编译器打开页面，会产生跨域问题')
26         console.log('地址为: http://localhost:3000/code.html')
27     }
28 })

```

## 2 跨域

### 1.为什么会有跨域这个问题？

原因是浏览器为了安全，而采用的同源策略（Same origin policy）

### 2.什么是同源策略？

1. 同源策略是由Netscape提出的一个著名的安全策略，现在所有支持JavaScript 的浏览器都会使用这个策略。

2. Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

3. 所谓同源是指：协议，域名（IP），端口必须要完全相同

即：协议、域名（IP）、端口都相同，才能算是在同一个域里。

### 3.非同源受到哪些限制？

1. Cookie不能读取；

2. DOM无法获得；

3. Ajax请求不能获取数据

### 4.如何在开发中解决跨域问题：

1.JSONP解决发送请求跨域问题：

要明确的是：JSONP不是一种技术，而是程序员“智慧的结晶”（利用了标签请求资源不受同源策略限制的特点）

JSONP需要前后端人员互相配合。

前端页面写法：

```
1 <body>
```



```

2   <button id="btn">按钮</button>
3   <script type="text/javascript">
4       var btn = document.getElementById('btn');
5       btn.onclick = function () {
6           //1. 创建一个script标签
7           var script = document.createElement('script');
8           //2. 设置回调函数
9           window.getData = function (data) {
10               console.log(data); //拿到数据
11           }
12           //3. 设置script标签src属性，填写跨域请求的地址
13           script.src = 'http://localhost:3000/jsonp?callback=getData';
14           //4. 将script标签添加到body中生效
15           document.body.appendChild(script);
16           //5. 不影响整体DOM结构，删除script标签
17           document.body.removeChild(script);
18       }
19   </script>
20 </body>

```

后端写法:

```

1 app.get('/jsonp', (req, res) => {
2     //解构赋值获取请求参数
3     const { callback } = req.query
4     //去数据库查找对应数据
5     const data = [{ name: 'tom', age: 18 }, { name: 'jerry', age: 20 }];
6     res.send(callback + '(' + JSON.stringify(data) + ')');
7 })

```

2.后台配置cors解决跨域\*\*

以Node为例:

res.set('Access-Control-Allow-Origin', 'http://localhost:63342');

\*\*3.使用代理服务器\*\*

例如: nginx等

### 3. JSONP解决跨域

```

1 <!DOCTYPE html>

```

```

2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Title</title>
7   <script type="text/javascript" src="./jquery.min.js"></script>
8 </head>
9 <body>
10  <button id="btn1">点我使用jquery封装的jsonp获取数据</button>
11
12  <script type="text/javascript">
13    let btn1 = $('#btn1')
14    btn1.click(() => {
15      //完整写法
16      /*$.ajax({
17        url:'http://localhost:3000/test_get',
18        method:'get',
19        dataType:'jsonp',
20        data:{name:'kobe',age:18},
21        success:(result)=>{
22          console.log(result)
23        },
24        error:(err)=>{
25          console.log(err)
26        }
27      })*//
28
29      //简单写法
30      $.getJSON('http://localhost:3000/test_get?callback=?', { name: 'kobe', age: 18 },
31        (data) => {
32          console.log(data)
33        })
34    })
35  </script>
36 </body>
37 </html>

```

```

1 <!DOCTYPE html>

```

```
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>原生js写jsonp</title>
7 </head>
8
9 <body>
10  <button id="btn">点我获取数据</button>
11  <form action="http://localhost:3000/test_get" method="get">
12    <input type="submit">
13  </form>
14
15  <script type="text/javascript">
16    /*
17     * jsonp解决跨域:
18     *   1.利用script标签, 发请求不受到同源策略的限制。
19     *   2.需要后端工程师配合
20     *   3.不可以解决post请求跨域的问题
21     * */
22
23    let btn = document.getElementById('btn')
24    btn.onclick = () => {
25      //1.创建一个script标签
26      let scriptNode = document.createElement('script')
27      //2.提前定义好一个函数
28      window.demo3 = function (data) {
29        console.log(data)
30      }
31      //3.为script标签指定src属性
32      scriptNode.src = 'http://localhost:3000/test_get?callback=demo3'
33      //4.将script标签加入页面
34      document.body.appendChild(scriptNode)
35    }
36  </script>
37
38 </body>
39 </html>
```

```

1  const express = require('express')
2
3  const app = express()
4
5  app.use(express.urlencoded({ extended: true }))
6
7  app.use(express.json())
8
9  app.use(express.static(__dirname + '/public'))
10
11 app.get('/test_get', (req, res) => {
12   const { callback } = req.query
13   console.log(callback)
14   let person = [{ name: 'kobe', age: 18 }, { name: 'kobe', age: 18 }, { name: 'kobe',
age: 18 }, { name: 'kobe', age: 18 }]
15   res.send(`${callback}(${JSON.stringify(person)})`)
16 })
17
18 app.listen(3000, (err) => {
19   if (err) console.log(err)
20   else {
21     console.log('兄弟必须要用编译器打开页面，你要解决跨域问题')
22   }
23 })

```

#### 4.jQuery封装ajax:

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>封装ajax</title>
7  </head>
8
9  <body>
10   <button id="btn">点我使用自己封装的ajax请求数据</button>
11   <script type="text/javascript">
12

```

```
13 let btn = document.getElementById('btn')
14 btn.onclick = () => {
15     sendAjax({
16         url: 'http://localhost:3000/test_post',
17         method: 'post',
18         data: { name: 'kobe', age: 18 },
19         success: (result) => {
20             console.log(result)
21         },
22         error: (err) => {
23             console.log(err)
24         }
25     })
26 }
27
28 //用于发送ajax请求的函数，封装的是原生xhr
29 function sendAjax(options) {
30
31     //1.从配置对象中获取发请求所需的参数
32     let { url, method, data, success, error } = options
33     method = method || 'get'
34
35     //2.实例化一个xhr
36     let xhr = new XMLHttpRequest()
37
38     //3.监听状态
39     xhr.onreadystatechange = () => {
40         if (xhr.readyState !== 4) return
41         if (xhr.readyState === 4 && (xhr.status >= 200 && xhr.status <= 299)) {
42             if (success) success(xhr.response)
43         } else {
44             if (error) error('请求失败')
45         }
46     }
47
48     //4.整理参数
49     let str = ''
50     if (data) {
51         for (let key in data) {
```

```

52         str += `${key}=${data[key]}&`
53     }
54 }
55
56 //5. 此处应该判断请求方式，根据请求的方式，决定如何携带参数
57 if (method.toUpperCase() === 'GET') {
58     xhr.open(method, `${url}?${str}`)
59     xhr.send()
60 } else {
61     xhr.open(method, url)
62     xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded')
63     xhr.send(str)
64 }
65
66 }
67 </script>
68
69 </body>
70
71 </html>

```

```

1  //引入express
2  const express = require('express')
3  //实例一个app应用对象
4  const app = express()
5  //使用内置中间件去解析post请求中以urlencoded形式编码的参数
6  app.use(express.urlencoded({ extended: true }))
7  app.use(express.json())
8  //暴露静态资源
9  app.use(express.static(__dirname + '/public'))
10
11 app.get('/test_get', (req, res) => {
12     res.set('Access-Control-Allow-Origin', '*')
13     console.log('有人发来了get请求')
14     console.log(req.query)
15     res.send('你发来的是get请求，我收到了，这是给你的数据：哈哈')
16 })
17

```

```
18 app.post('/test_post', (req, res) => {
19   res.set('Access-Control-Allow-Origin', '*')
20   console.log(req.body);
21   res.send('你发来的是post请求，我收到了，这是给你的数据：哈哈')
22 })
23
24 app.listen(3000, (err) => {
25   if (err) console.log(err)
26   else {
27     console.log('服务器启动成功')
28   }
29 })
```

